# Preprocessing Argumentation Frameworks via Replacement Patterns

Wolfgang Dvořák[1 [0000−0002−2269−8193]], Matti Järvisalo[2 [0000−0003−2572−063X]], Thomas Linsbichler[1 [0000−0001−9143−4071]], Andreas Niskanen[2 [0000−0003−3197−2075]], and Stefan Woltran[1 [0000−0003−1594−8972]]

[1] Institute of Logic and Computation, TU Wien, Austria
[2] HIIT, Department of Computer Science, University of Helsinki, Finland

**Abstract.** A fast-growing research direction in the study of formal argumentation is the development of practical systems for central reasoning problems underlying argumentation. In particular, numerous systems for abstract argumentation frameworks (AF solvers) are available today, covering several argumentation semantics and reasoning tasks. Instead of proposing another algorithmic approach for AF solving, we introduce in this paper distinct AF preprocessing techniques as a solver-independent approach to obtaining performance improvements of AF solvers. We establish a formal framework of replacement patterns to perform local simplifications that are faithful with respect to standard semantics for AFs. Moreover, we provide a collection of concrete replacement patterns. Towards potential applicability, we employ the patterns in a preliminary empirical evaluation of their influence on AF solver performance.

**Keywords:** abstract argumentation · preprocessing · extension enumeration

## 1 Introduction

Argumentation is today a vibrant area of modern AI research [4]. In particular, the study of computational aspects of argumentation connects with several AI subfields such as knowledge representation, constraints, and complexity of reasoning. The development of practical systems and algorithmic solutions for central reasoning problems underlying argumentation is motivated by a range of applications [1].

Abstract argumentation offers argumentation frameworks (AFs) as an important graph-based knowledge representation formalism for argumentation [9]. Computational models of argumentation, in particular from the perspective of the development of practical algorithms, have recently received a lot of attention. Several optimized practical AF reasoning systems (AF solvers) are available today [6], covering several argumentation semantics and reasoning tasks such as enumeration and skeptical and credulous query answering. The various state-of-the-art AF solvers, developed by several research groups around the world, are evaluated in the biennially organized ICCMA AF solver competitions [22,13], providing further incentives for seeking improvements in AF solver technology.

While both specialized and constraint-based AF solvers have been developed, less attention has been so far put on the development of preprocessing and simplification

techniques working directly on AFs. This is despite the fact that polynomial-time preprocessing (rewriting) has been shown to bring great practical performance improvements in various constraint solving paradigms [11,18,17,20,14,19]. Notably, preprocessing techniques applied before invoking a solver for the reasoning task at hand are *solver-independent*. Thereby the development of practical preprocessing techniques has the potential of improving the performance of various solvers. As proven to be the case in the area of propositional reasoning [18], applying combinations of relatively simple individual preprocessing techniques can have a surprisingly significant positive effect on solver performance.

In this work, we take first steps in solver-independent preprocessing for AFs. By preprocessing we understand a family of polynomial-time applicable simplification rules which preserve an appropriate form of equivalence. In the domain of AFs this amounts to searching for particular sub-AFs that can be replaced by a smaller and/or simpler AF without changing the semantics of the whole AF. However, the nonmonotonic nature of AF semantics makes the understanding of such replacements non-trivial. In addition, we aim for removing arguments that cannot be accepted and for merging arguments that can only be jointly accepted. Since preprocessing itself should rely on efficient polynomial-time algorithms, we cannot include any semantic treatment of AFs or sub-AFs into the procedures.

To this end, we introduce the concept of *replacement patterns* which contain information about (i) which AFs need to be matched on subgraphs of the AF at hand, and (ii) how to simplify them independently of the surrounding framework. The recently introduced notion of *$C$-relativized equivalence* [5] provides a suitable tool to prove faithfulness of such simplifications. However, we need to extend this notion properly in order to also capture the concept of merging of arguments in a formally sound way. Our formal results refine equivalence results for AFs with merged arguments and show how these can be used to show faithfulness of our patterns. Consequently, AFs obtained by iterative applications of replacement patterns are equivalent to the original AF one started with, which makes them applicable even for the task of extension enumeration.

An alternative approach to local simplifications of AFs is the S-equivalence of multipoles [2]. However, S-equivalence treats the part of the AF to be replaced as "black-box" and thus allows for changes in the extensions w.r.t. arguments which are not in the IO-interface of the multipole. As our work is focusing on replacements that preserve the extensions of the AF we thus follow and extend the approach of [5].

The preprocessed AFs obtained via applications of replacement patterns can be input to any state-of-the-art AF solver for obtaining solutions to enumeration and reasoning tasks over the original input AF. After obtaining a solution to the preprocessed AF, the only additional tasks to be performed is to reconstruct actual arguments from merged arguments, which is straightforward for our replacement patterns.

We provide a set of concrete polynomial-time checkable replacement patterns which we consider as a first suite of solver-independent AF preprocessing techniques for stable, preferred, and complete semantics. We further study the impact of our preprocessing routine via a preliminary empirical evaluation on both two state-of-the-art native AF solvers [15,21] and a SAT-based AF solver [10,12] on the task of extension enumeration. Our results reveal that in particular the native solvers can highly benefit from pre-

processing; also the performance of constraint-based solvers can be improved at times. Hence preprocessing appears promising for further closing the current performance gap [8] between state-of-the-art native and constraint-based solvers, and the first empirical results presented motivate further studies of practical preprocessing techniques for different AF reasoning tasks, including acceptance problems where preprocessing needs not to preserve all extensions, in constrast to extension enumeration.

The paper is organized as follows. We first recall abstract argumentation and the notion of *$C$-relativized equivalence* (Section 2). Next, we introduce *replacement patterns* as a formal framework for studying faithful simplifications of AFs, and extend the notion of $C$-relativized equivalence to allow for formally establishing faithfulness of replacement patterns (Section 3.1). We then provide concrete replacement patterns for preprocessing AFs (Section 3.2). Finally, we present an empirical evaluation of the presented patterns (Section 4).

## 2 Argumentation Frameworks and Equivalence

We recall abstract argumentation frameworks [9], their semantics (see [3] for an overview), and the notion of $C$-relativized equivalence [5] which we will employ to show the faithfulness of replacements. We fix $U$ as a countably infinite domain of arguments.

**Definition 1.** *An* argumentation framework (AF) *is a pair $F = (A, R)$ where $A \subseteq U$ is a finite set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that $a$ attacks $b$. We use $A_F$ to refer to $A$ and $R_F$ to refer to $R$. We say that an AF is given over a set $B$ of arguments if $A_F \subseteq B$.*

**Definition 2.** *Given an AF $F$ and set $S \subseteq A_F$, we define $S^+_F = \{x \mid \exists y \in S : (y, x) \in R_F\}$, $S^-_F = \{x \mid \exists y \in S : (x, y) \in R_F\}$, and the* range of $S$ *in $F$ as $S^\oplus_F = S \cup S^+_F$.*

The following adaptions of set-theoretic operators to pairs of AFs will be useful in the rest of the paper.

**Definition 3.** *Given AFs $F = (A, R)$, $F' = (A', R')$, we denote the* union of AFs *as $F \cup F' = (A \cup A', R \cup R')$. For a set $S \subseteq U$ of arguments, and a set $T \subseteq (A \times A)$ of attacks, we define $F \setminus S = (A \setminus S, R \cap ((A \setminus S) \times (A \setminus S)))$, $F \setminus T = (A, R \setminus T)$, $F \cap S = (A \cap S, R \cap ((A \cap S) \times (A \cap S)))$, $F \cup S = (A \cup S, R)$, and $F \cup T = (A, R \cup T)$. For mixed sets $S \cup T$ of arguments $S$ and attacks $T$ we define $F \setminus (S \cup T) = (F \setminus T) \setminus S$,*

We next give a formal definition of sub-AFs. In words, a sub-AF is an induced subgraph of the directed graph representation of an AF.

**Definition 4.** *We call an AF $F$ to be a* sub-AF *of $G$, in symbols $F \sqsubseteq G$, if $A_F \subseteq A_G$ and $R_F = R_G \cap (A_F \times A_F)$.*

Replacement is a central notion in this work, and intuitively defines substitutions of a sub-AF with another.

**Definition 5.** *Given AFs $F, F', G$ such that $F \sqsubseteq G$ and $A_{F'} \cap (A_G \setminus A_F) = \emptyset$, let $A = (A_G \setminus A_F) \cup A_{F'}$. The* replacement of $F$ by $F'$ in $G$ *is defined as $G[F/F'] = (A, ((R_G \setminus R_F) \cap (A \times A)) \cup R_{F'})$.*

Semantics for AFs are defined based on the notions of defense and the characteristic function.

**Definition 6.** *Given an AF $F = (A, R)$, argument $a \in A$ is* defended *(in $F$) by a set $S \subseteq A$ if $\{a\}_F^- \subseteq S_F^+$. The* characteristic function $\mathcal{F}_F : 2^A \to 2^A$ *of $F$ is defined as $\mathcal{F}_F(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } F\}$.*

Semantics are functions $\sigma$ which assign to each AF $F$ a set $\sigma(F) \subseteq 2^{A_F}$ of extensions. We consider for $\sigma$ the functions *stb*, *com*, and *prf*, which stand for stable, complete, and preferred semantics, respectively.

**Definition 7.** *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in $F$)*, if there are no $a, b \in S$ such that $(a, b) \in R$. cf($F$) denotes the collection of conflict-free sets of $F$. For a conflict-free set $S \in$ cf($F$), it holds that*

- *$S \in$ stb($F$), if $S_F^\oplus = A$;*
- *$S \in$ com($F$), if $S = \mathcal{F}_F(S)$;*
- *$S \in$ prf($F$), if $S \in$ com($F$) and $\nexists T \supset S$ s.t. $T \in$ com($F$).*

Under a standard notion, two AFs are equivalent under a semantics iff they have the same extensions.

**Definition 8.** *We call two AFs $F$ and $G$ to be equivalent in semantics $\sigma$, in symbols $F \equiv^\sigma G$, iff $\sigma(F) = \sigma(G)$.*

Baumann et al. [5] have studied the following refined notion of equivalence, which is sensitive to expansions as long as they do not affect a certain core of arguments.

**Definition 9.** *Given a semantics $\sigma$ and $C \subseteq U$. Two AFs $F$ and $G$ are* $C$-relativized equivalent *w.r.t. $\sigma$ ($F \equiv_C^\sigma G$) iff $F \cup H \equiv^\sigma G \cup H$ holds for each AF $H$ over $U \setminus C$.*

In order to decide whether two AFs are $C$-relativized equivalent, $C$-restricted semantics have been introduced. Those restrict the relevant properties of the original semantics to the core arguments $C$. We present these concepts in detail for stable semantics, on which we shall focus in the remainder of the paper.

**Definition 10.** *Let $F$ be an AF, $C \subseteq U$ and $E \subseteq A_F$. We have that $E \in stb_C(F)$ if $E \in$ cf($F$) and $A_F \cap C \subseteq E_F^\oplus$.*

That is, for $C$-restricted stable extensions we relax the conditions for stable semantics such that, beside the extension being conflict-free, we only require that all arguments in the core are in the range of the extension.

We have that two AFs $F, G$ can only be $C$-relativized equivalent w.r.t. a semantics $\sigma$ if they have exactly the same $C$-restricted $\sigma$ extensions. However, this is only a necessary but not a sufficient condition. We additionally require that, except for arguments in the core $C$, the AFs $F, G$ have the same arguments and for stable semantics that all $C$-restricted stable extensions have the same range in $F$ and $G$ when ignoring the core arguments $C$.

**Theorem 1.** *[5] Let $F, G$ be AFs and $C \subseteq U$. Then, $F \equiv_C^{stb} G$ iff the following jointly hold:*

1. *if $stb_C(F) \neq \emptyset$, $A_F \setminus C = A_G \setminus C$;*
2. *$stb_C(F) = stb_C(G)$; and*
3. *for all $E \in stb_C(F)$, $E_F^+ \setminus C = E_G^+ \setminus C$.*

## 3 Replacement Patterns

The idea behind the notion of replacement patterns is to allow for some freedom in the subgraphs we are looking for to apply simplifications. For example, often the existence of certain attacks does not affect the applicability of replacements; a specific replacement pattern defines similar graphs that qualify for replacements. In what follows, we first introduce the formal framework of replacement patterns and show how faithfulness of patterns can be achieved using the notion of $C$-relativized equivalence. Then we present concrete replacement patterns for preprocessing AFs.

### 3.1 Main Concepts

A central ingredient of replacement patterns is merging of arguments, resulting in arguments of the form $m_S$ with $S \subseteq U$ being standard arguments. The universe of all such arguments is given by $U_m = \{m_S \mid S \subseteq U, S \text{ is finite}\}$.

**Definition 11.** *Let $F = (A, R)$ be an AF and $a, b \in A$. The merge $\mathsf{M}(F, a, b)$ of $a, b$ in $F$ is the AF $(A', R')$ given by $A' = A \setminus \{a, b\} \cup \{m_{\{a,b\}}\}$ [3] and $R' = R \cap (A' \times A') \cup \{(m_{\{a,b\}}, c) \mid (a, c) \in R \text{ or } (b, c) \in R\} \cup \{(c, m_{\{a,b\}}) \mid (c, a) \in R \text{ or } (c, b) \in R\}$.*

The following two unpacking functions $\mathsf{U}(\cdot)$ map (i) a set of arguments over $U \cup U_m$ to the corresponding set of arguments in $U$, and (ii) an AF with merged arguments back to an AF over $U$.

**Definition 12.** *Let $F = (A, R)$ be an AF with $A \subseteq U \cup U_m$ and $E \subseteq A$. The unpacked set $\mathsf{U}(E)$ of $E$ is given by $(E \cap U) \cup \bigcup_{m_S \in E} S$. The unpacked AF $\mathsf{U}(F)$ of $F$ is the AF $(A', R')$ given by $A' = \mathsf{U}(A)$ and $R' = R \cap (A' \times A') \cup \{(a, c) \mid (m_S, c) \in R, a \in S\} \cup \{(c, a) \mid (c, m_S) \in R, a \in S\} \cup \{(a, c) \mid (m_S, m_{S'}) \in R, a \in S, c \in S'\}$.*

Notice that $\mathsf{U}(F)$ is always an AF over $U$. In the next step we generalize standard equivalence by taking into account AFs which have resulted from merging of arguments. That is we do not compare extensions directly but consider AFs to be equivalent if their unpacked extensions coincide.

**Definition 13.** *We call two AFs $F$ and $G$ over $U \cup U_m$ to be equivalent in semantics $\sigma$, in symbols $F \equiv^{\sigma} G$, iff $\{\mathsf{U}(E) \mid E \in \sigma(F)\} = \{\mathsf{U}(E) \mid E \in \sigma(G)\}$.*

We are now ready to give a formal notion of a replacement pattern. In order to define a replacement for a class of (similar) graphs instead of just a single graph in our replacement pattern we have to define a replacement for each of the graphs in the class. That is, a replacement pattern $P_C$ consists of pairs $(F, F')$ that coincide on arguments not in $C$. When applying such a pattern $P_C$ to a larger AF $G$ some sub-AF $M$ of $G$ that is isomorphic to $F$ is replaced by a graph isomorphic to $F'$. We first give a formal definition of replacement patterns and then define how to apply such patterns to AFs.

[3] However, we keep the set structure flat, i.e., when merging arguments $m_S, m_{S'} \in U_m$ the resulting argument is $m_{S \cup S'}$.

**Definition 14 (Replacement pattern).** *A replacement pattern $P_C$ for $C \subseteq U$ is a set of pairs $(F, F')$ of AFs $F, F'$ such that $A_F \subseteq U$, $A_{F'} \subseteq U \cup U_m$, and $F$ and $F'$ coincide on the arguments not contained in $C_m = C \cup \{m_S \mid S \subseteq C\}$, i.e., a replacement pattern is of the form*

$$P_C = \{(F, F') \mid F, F' \text{ AFs}, F \setminus C = F' \setminus C_m\},$$

*such that for any $(F_1, F_1'), (F_2, F_2') \in P_C$, $F_1 \neq F_2$.*

For preprocessing, we need to detect an instantiation of a pattern $P_C$ as what we call $C$-encircling sub-AF of an AF $G$, i.e. a sub-AF $F$ such that $C$ might be connected to $F$ but is not connected to $G \setminus F$, and then apply the pattern in the form of a replacement.

**Definition 15 ($C$-encircling sub-AF).** *An AF $F$ is a* $C$-encircling sub-AF *of an AF $G$ if (i) $F \sqsubseteq G$ and (ii) $C_G^{\oplus} \cup C_G^{-} \subseteq A_F$, i.e., $C$ is not connected to $A_{G \setminus F}$ in $G$.*

Now a match of a pattern $P_C = \{(F_i, F_i') \mid 1 \leq i \leq k\}$ on $G$ is a $C'$-encircling sub-AF $I$ of $G$ that is isomorphic to some $F_i$ where $C'$ is the image of $C$ under the isomorphism from $F_i$ to $I$.

**Definition 16 (Applying $P_C$).** *Given AF $G$ and pattern $P_C$, a* match *of $P_C$ on $G$ is a tuple $(F, F', I, \alpha)$, where $(F, F') \in P_C$, $I$ is a $C'$-encircling sub-AF of $G$, and $I$ is isomorphic to $F$ via isomorphism $\alpha : A_F \to A_G$ such that $\alpha(C) = C'$. We say a pattern $P_C$* can be applied to *$G$ if there exists a match of $P_C$ on $G$. An application $P_C[G]$ of pattern $P_C$ on $G$ then picks a match $(F, F', I, \alpha)$ of $P_C$ on $G$ and returns $G[I/\alpha(F')]$, where $\alpha$ is extended to arguments $m_S$ by mapping them to $m_{\alpha(S)}$.*

The following example illustrates these concepts.

*Example 1.* Consider the replacement pattern $P^Y_{\{a,b,c\}}$ containing the pair $(F, F')$ with $F = (\{a, b, c, d, e\}, \{(d, a), (a, b), (b, c), (c, e), (e, d)\})$ and $F' = \mathsf{M}(F \setminus \{(b, c)\}, a, c) = (\{m_{\{a,c\}}, b, d, e\}, \{(d, m_{\{a,c\}}),\ (m_{\{a,c\}}, b), (m_{\{a,c\}}, e), (e, d)\})$. Moreover, consider the AF $G$ depicted in in Figure 1 (left). Now observe that the tuple $(F, F', I, \alpha)$ is a match of $P^Y_{\{a,b,c\}}$ on $G$ with $I = G \cap \{x_1, \ldots, x_5\}$ and $\alpha = \{a \mapsto x_1, b \mapsto x_2, c \mapsto x_3, d \mapsto x_5, e \mapsto x_4\}$. Hence $P^Y_{\{a,b,c\}}$ can be applied to $G$, resulting in the AF $G[I/\alpha(F')] = G'$ depicted in Figure 1 (center). For stable semantics, we can verify that this replacement is equivalence preserving, since $\mathit{stb}(G) = \{\{x_0, x_1, x_3\}\}$ and $\mathit{stb}(G') = \{\{x_0, m_{\{x_1,x_3\}}\}\}$, meaning that $\{\mathsf{U}(E) \mid E \in \mathit{stb}(G)\} = \{\mathsf{U}(E) \mid E \in \mathit{stb}(G')\} = \{\{x_0, x_1, x_3\}\}$. Note, however, that $\mathsf{U}(G') \neq G$, since $\mathsf{U}(G')$ contains the attacks $(x_3, x_2)$, $(x_5, x_3)$, and $(x_1, x_4)$, which are not present in $G$. ◊

Naturally, a replacement pattern is faithful only if each of its possible applications is an equivalence-preserving modification.

**Definition 17 (Faithful pattern).** *A replacement pattern $P_C$ is $\sigma$-faithful iff $P_C[G] \equiv^{\sigma} G$ for all $G$ over $U \cup U_m$.*

Testing whether a replacement pattern is faithful can be reduced to testing $C$-relativized equivalence of the pairs of AFs covered by the pattern. This applies directly to patterns that do not involve the merging of arguments, and requires unpacking for patterns that do.

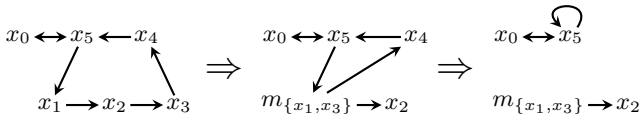


**Fig. 1.** Applying (i) the 3-path pattern (cf. Example 1) and (ii) the 3-loop pattern (cf. Example 3).

**Theorem 2.** *For semantics $\sigma \in \{stb, prf, com\}$ and replacement pattern $P_C$ such that for each $(F, F') \in P_C$, $A_{F'} \cap S = \emptyset$ for $m_S \in A_{F'}$ and $S \cap S' = \emptyset$ for $m_S, m_{S'} \in A_{F'}$, the following statements are equivalent.*

1. *$P_C$ is $\sigma$-faithful.*
2. *$F \equiv_C^\sigma \mathsf{U}(F')$ for each $(F, F') \in P_C$.*

We next continue our example to illustrate how one can use the above theorem in order to prove our pattern to be *stb*-faithful.

*Example 2.* Again consider the replacement pattern $P_C^Y$ with $C = \{a, b, c\}$ from Example 1 and assume it contains just the pair $(F, F')$. Now observe that $\mathsf{U}(F') = F \setminus \{(b, c)\} \cup \{(a, e), (d, c), (c, b)\}$. It holds that (1) $A_F \setminus C = A_{\mathsf{U}(F')} \setminus C = \{d, e\}$, (2) $stb_C(F) = stb_C(\mathsf{U}(F')) = \{\{a, c\}, \{b, d\}\}$, and (3) $\{a, c\}_F^+ \setminus C = \{a, c\}_{\mathsf{U}(F')}^+ \setminus C = \{e\}$ and $\{b, d\}_F^+ \setminus C = \{b, d\}_{\mathsf{U}(F')}^+ \setminus C = \emptyset$. It thus holds that $F \equiv_C^{stb} \mathsf{U}(F')$ (cf. Theorem 1) and, by Theorem 2, that $P_C^Y$ is *stb*-faithful. $\Diamond$

**Proof of Theorem 2**

In this section we provide a proof of Theorem 2. We call an AF that meets the conditions of Theorem 2 an *arg-unique* AF.

**Definition 18.** *An AF $F$ over $U \cup U_m$ is called* arg-unique *if $A_F \cap S = \emptyset$ for $m_S \in A_F$ and $S \cap S' = \emptyset$ for $m_S, m_{S'} \in A_{F'}$.*

We first observe that two arg-unique AFs are equivalent iff their unpackings are equivalent. The proof of the lemma exploits the fact that the unpacked extensions of an arg-unique AF $F$ coincide with the extensions of its unpacking $\mathsf{U}(F)$.

**Lemma 1.** *For semantics $\sigma \in \{stb, prf, com\}$, arg-unique AFs $F, G$ over $U \cup U_m$ we have $F \equiv^\sigma G$ iff $\mathsf{U}(F) \equiv^\sigma \mathsf{U}(G)$.*

*Proof.* We show that $\sigma(\mathsf{U}(F)) = \{\mathsf{U}(E) \mid E \in \sigma(F)\}$ which implies the lemma.

$\subseteq$: Consider $E \in \sigma(\mathsf{U}(F))$ and an argument $m_S \in A_F \cap U_m$. As all arguments in $S$ have the same attackers in $\mathsf{U}(F)$ we have that either $S \subset E$ or $S \cap E = \emptyset$. Now it is easy to verify that for the set $E' = \{a \mid a \in A_F \cap E\} \cup \{m_S \mid m_s \in A_F, S \subseteq E\}$ it holds that $E' \in \sigma(F)$ and $\mathsf{U}(E') = E$.

$\supseteq$: For $E \in \sigma(F)$ it is easy to verify that $\mathsf{U}(E) \in \sigma(\mathsf{U}(F))$. In particular, $m_S$ is defended by $E$ in $F$ iff each $a \in S$ is defended by $\mathsf{U}(E)$ in $\mathsf{U}(F)$. $\square$

We next extend the notion of $C$-relativized equivalence to AFs over $U \cup U_m$.

**Definition 19.** *Given a semantics $\sigma$ and $C \subseteq U \cup U_m$. Two AFs $F$ and $G$ over $U \cup U_m$ are* $C$-relativized equivalent *w.r.t. $\sigma$ ($F \equiv^\sigma_C G$) iff $F \cup H \equiv^\sigma G \cup H$ for $H$ over $(U \cup U_m) \setminus C$.*

Notice that if $F, G$ are AFs over $U$ the above notion coincides with the earlier notion of $C$-relativized equivalence. We next show that a pattern with core $C$ is faithful iff the two AFs in each of the pattern's pairs are $C_m$-relativized equivalent.

**Proposition 1.** *For $C \subseteq U$ and $C_m = C \cup \{m_S \mid S \subseteq C\}$ the pattern $P_C$ is $\sigma$-faithful iff $F \equiv^\sigma_{C_m} F'$ for each $(F, F') \in P_C$.*

*Proof.* 2 $\Rightarrow$ 1) We have to show $P_C[G] \equiv^\sigma G$, for every $G$ and every possible match. Consider a match $(F, F', I, \alpha)$. First notice that $F \equiv^\sigma_{C_m} F'$ implies $I = \alpha(F) \equiv^\sigma_{C'} \alpha(F') = I'$ (with $C' = \alpha(C)$) as the equivalence does not depend on the names of arguments from $U$, but only on whether they are in the core $C$, resp. $C'$, which is maintained by $\alpha$. By the definition of $\equiv^\sigma_{C'}$ we have $I \cup H \equiv^\sigma I' \cup H$ for all $H$ that do not contain arguments from $C'_m$. Finally, by setting $H = G \setminus C'$ we obtain that $P_C[G] \equiv^\sigma G$.

1 $\Rightarrow$ 2) If $F \not\equiv^\sigma_C F'$ for some $(F, F') \in P_C$ there is an AF $H$ over $(U \cup U_m) \setminus (C \cup \{m_S \mid S \subseteq C\}$ such that $F \cup H \not\equiv^\sigma F' \cup H$. Now as there is a match with $P_C[F \cup H] = F' \cup H$ we obtain that $P_C$ is not $\sigma$-faithful. □

Finally, for arg-unique AFs the $C$-relativized equivalence tests for $(F, F') \in P_C$ can be reduced to $C$-relativized equivalence tests on AFs over $U$. That is, to the case already studied and well characterised in [5].

**Lemma 2.** *For semantics $\sigma \in \{stb, prf, com\}$, cores $C \subseteq U$, $C_m = C \cup \{m_S \mid S \subseteq C\}$, AF $F$ over $U$ and arg-unique AF $F'$ over $U \cup U_m$ such that $F \setminus C = F' \setminus C_m$, the following statements are equivalent:*

1. $F \equiv^\sigma_{C_m} F'$.
2. $F \equiv^\sigma_C \mathsf{U}(F')$.

The proof of the lemma is based on the observation that given an AF $H$ over $U \cup U_m$ such that $F \not\equiv^\sigma_{C_m} F'$, by exploiting Lemma 1 we can construct an AF $H'$ over $U$ showing $F \not\equiv^\sigma_C \mathsf{U}(F')$, and vice versa.

*Proof.* 2 $\Rightarrow$ 1) W.l.o.g. assume there are $E$ and $H$ such that $E \in \sigma(F \cup H)$ but $E \notin \sigma(F' \cup H)$. It is easy to verify that $\mathsf{U}(F' \cup H) = \mathsf{U}(F') \cup \mathsf{U}(H)$ (notice that $F'_A \cap H_A \subseteq F_A \subseteq U$). By Lemma 1 we have $\mathsf{U}(F \cup H) \not\equiv^\sigma \mathsf{U}(F' \cup H)$. Thus there is an $E'$ such that $E' \in \sigma(F \cup \mathsf{U}(H))$ but $E' \notin \sigma(\mathsf{U}(F') \cup \mathsf{U}(H))$, i.e., $\mathsf{U}(F) \cup \mathsf{U}(H) \not\equiv^\sigma \mathsf{U}(F') \cup \mathsf{U}(H)$. Moreover, by construction, $\mathsf{U}(H)$ does not contain arguments from $\mathsf{U}(C)$. Hence, $\mathsf{U}(F) \not\equiv^\sigma_{\mathsf{U}(C)} \mathsf{U}(F')$.

1 $\Rightarrow$ 2) W.l.o.g. assume there is a set $E$ and an AF $H$ such that $E \in \sigma(F \cup H)$ but $E \notin \sigma(\mathsf{U}(F') \cup H)$. As $A_F \setminus C = A_{F'} \setminus C_m$ the set $A_{F'} \setminus C_m$ does not contain merged arguments. Thus we have $\mathsf{U}(F') \cup H = \mathsf{U}(F' \cup H)$ and, by Lemma 1, we have $E \in \sigma(F \cup H)$ but $E \notin \sigma(\mathsf{U}(F') \cup H)$, i.e., $F \cup H \not\equiv F' \cup H$. Hence, $F \not\equiv^\sigma_{C_m} F'$. □

Finally, Theorem 2 is immediate by combining Proposition 1 with Lemma 2.

### 3.2 Formalizing Concrete Patterns

We will now present our concrete replacement patterns. For this, we will use the concept of a *lagging*. Intuitively, for a given core-AF $F$ a lagging $F_L$ is an AF extending $F$ by new arguments that either attack or are attacked by arguments in $F$. When defining our patterns we are typically interested in all laggings of a specific core-AF satisfying certain conditions.

**Definition 20.** *Given an AF $F = (A, R)$, a* lagging *of $F$ is any AF $F_L = (A', R')$ with $A \subseteq A'$ such that $F_L \cap A = F$ and $A^{\oplus}_{F_L} \cup A^{-}_{F_L} = A'$. Given a lagging $F_L$, we sometimes refer to $F$ as the core-AF.*

For instance, the AF $F$ in the pattern of Example 1 is a lagging of the AF $(\{a, b, c\}, \{(a, b), (b, c)\})$.

We have already given a glimpse on one of the patterns in Examples 1 and 2. There, the pattern contained just a single pair of AFs $(F, F')$, where the core contained the directed path $a \to b \to c$. The insight that, given that $b$ and $c$ are otherwise unattacked, in such cases $a$ and $c$ can be merged as they will appear together in every (stable) extension, is the central concept to the following pattern.

**Definition 21.** *Let $F_{3P} = (\{a, b, c\}, \{(a, b), (b, c)\})$ be the core-AF. The* 3-path pattern *is given by*

$$P^{3P}_{\{a,b,c\}} = \{(F, F') \mid F \text{ is a lagging of } F_{3P}, \{b, c\}^{-}_{F} = \{a, b\}, F' = \mathsf{M}(F \setminus \{(b, c)\}, a, c)\}.$$

In words, the 3-path pattern concerns all AFs which contain a proper 3-path $a \to b \to c$ such that each argument $x$ different from $a, b, c$ is adjacent to this 3-path. In order to contain the 3-path properly, $x$ can only attack $a$, but it can be attacked by $a$, $b$, or $c$. Each such AF $F$ is replaced by merging $a$ and $c$, without taking $b \to c$ into account. Loosely speaking we aim to replace in $F$ the path $a \to b \to c$ by $m_{\{a,c\}} \to b$.

**Proposition 2.** *$P^{3P}_{\{a,b,c\}}$ is a stb-faithful replacement pattern.*

*Proof.* Due to Theorem 2 it suffices to show that $F \equiv^{\sigma}_{C} \mathsf{U}(F')$ holds for each $(F, F') \in P^{3P}_{C}$, where $C = \{a, b, c\}$. Consider an arbitrary $(F, F') \in P^{3P}_{C}$. First note that, since $F' = \mathsf{M}(F \setminus \{(b, c)\}, a, c)$, $\mathsf{U}(F') = (A_F, R_F \setminus \{(b, c)\} \cup \{(a, x) \mid (c, x) \in R_F\} \cup \{(c, x) \mid (a, x) \in R_F\} \cup \{(x, c) \mid (x, a) \in R_F\}$. Let $G = \mathsf{U}(F')$.

For $F \equiv^{\sigma}_{C} G$ we need to show that (1) if $\mathit{stb}_C(F) \neq \emptyset$ then $A_F \setminus C = A_G \setminus C$, (2) $\mathit{stb}_C(F) = \mathit{stb}_C(G)$, and (3) for all $E \in \mathit{stb}_C(F)$, $E^{+}_{F} \setminus C = E^{+}_{G} \setminus C$.

(1) is immediate by $A_G = A_F$. For (2), consider an arbitrary $E \in \mathit{stb}_C(F)$. By $F \cap C = F_{3P}$ ($F$ is a lagging of $F_{3P}$) and $\{b, c\}^{-}_{F} = \{a, b\}$, we have $\{b\}^{-}_{F} = \{a\}$. Hence either (i) $a \in E$ or (ii) $b \in E$. In case of (i) we get, since $\{c\}^{-}_{F} = \{b\}$ and $a$ attacks $b$, that also $c \in E$. As attacks among arguments $A_F \setminus C$ remain unchanged in $G$, i.e. $R_F \setminus (C \times C) = R_G \setminus (C \times C)$, we get that $E \in \mathit{cf}(G)$. As $(a, b) \in R_G$, also $E \in \mathit{stb}_C(G)$. In case of (ii) there must be some $x \in E$ with $(x, a) \in R_F$. By construction of $G$, then $(x, a), (x, c) \in R_G$, hence $E^{+}_{G} \supseteq C$. Consequently, $E \in$

$\mathit{stb}_C(G)$. Hence $\mathit{stb}_C(F) \subseteq \mathit{stb}_C(G)$. For the other direction, consider an arbitrary $E \in \mathit{stb}_C(G)$. By the same reason as above, either (i) $a \in E$ or (ii) $b \in E$. For (i) observe that, for each $x \in A_G$, $(x, c) \in R_G$ iff $(x, a) \in R_G$. Hence also $c \in E$. By $R_F \setminus (C \times C) = R_G \setminus (C \times C)$ and $(a, b) \in A_F$, it follows that $E \in \mathit{stb}_C(F)$. In case of (ii) there must be some $x \in E$ with $(x, a) \in R_G$, hence also $(x, a) \in R_F$. Moreover, $(b, c) \in R_F$, hence $E \in \mathit{stb}_C(F)$.

For (3) let $E \in \mathit{stb}_C(F)$. As before, we can distinguish between (i) $a, c \in E$ and (ii) $b \in E$. Now by construction of $G$ it holds that $S_F^+ \setminus C = S_G^+ \setminus C$ for any $S \supseteq \{a, c\}$, in particular for $E$. Also in case of (ii) we get $E_F^+ \setminus C = E_G^+ \setminus C$ since $a, c \notin E$.

We can conclude that $F \equiv_C^{\mathit{stb}} G$ and $P_C^{3P}$ is *stb*-faithful. □

Another candidate for simplification are odd-length cycles. More concretely, in every occurrence of a directed cycle of length 3, $a \to b \to c \to a$, where only $a$ is attacked from the outside, one can disregard $c$ as well as the attack $(a, b)$ when adding a self-loop to $a$. This is formalized in the following replacement pattern.

**Definition 22.** *Let* $F_{3L} = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$ *be the core-AF. The* 3-loop pattern *is given by*

$$P^{3L}_{\{a,b,c\}} = \{(F, F') \mid F \setminus \{(a, a), (c, c)\} \textit{ is a lagging of } F_{3L}, \{b, c\}_F^- \subseteq \{a, b, c\}, F' = (F \setminus \{c, (a, b)\}) \cup \{(a, a)\}\}.$$

*Example 3.* Consider the AF $G'$ in Figure 1 (center), which we obtained through application of $P^{3P}_{\{a,b,c\}}$ (cf. Example 1). We can now apply $P^{3L}_{\{a,b,c\}}$ on $G'$: the tuple $(F, F', I, \alpha)$ is a match of $P^{3L}_{\{a,b,c\}}$ on $G'$ with $F = F_{3L} \cup (\{a, b, d, e\}, \{(a, d), (d, a), (b, e)\})$ (a lagging of $F_{3L}$), $F' = F \setminus \{c, (a, b)\} \cup \{(a, a)\}$, $I = G'$, and $\alpha = \{a \mapsto x_5, b \mapsto m_{\{x_1,x_3\}}, c \mapsto x_4, d \mapsto x_0, e \mapsto x_2\}$. One can check that $(F, F') \in P^{3L}_{\{a,b,c\}}$ and $I$ is isomorphic to $F$ via $\alpha$. The result is the AF $G''$ depicted in Figure 1 (right). It holds that $G' \equiv^{\mathit{stb}} G''$ since $\{\mathsf{U}(E) \mid E \in \mathit{stb}(G')\} = \{\mathsf{U}(E) \mid E \in \mathit{stb}(G'')\} = \{\{x_0, x_1, x_3\}\}$. ◊

Now consider two arguments in arbitrary attack relation, one of them being otherwise unattacked. Then, any stable extension must contain one of the two arguments. Hence, under *stb*, we can safely remove any argument that is attacked by the two without attacking back, together with all incident attacks. The following pattern expresses this simplification.

**Definition 23.** *Let* $F_{3C} = (\{a_1, a_2, b\}, \{(a_1, b), (a_2, b)\})$ *be the core-AF. The* 3-cone pattern *is given by*

$$P^{3C}_{\{a_1,a_2,b\}} = \{(F, F') \mid F \setminus \{(a_1, a_2), (a_2, a_1)\} \textit{ is a lagging of } F_{3C}, \{a_2\}_F^- \subseteq \{a_1\}, F' = F \setminus \{b\}\}.$$

The pattern $P^{3C}_{\{a_1,a_2,b\}}$ is illustrated in Figure 2. Solid edges represent necessary attacks while optional attacks are given by dotted edges.

The next pattern expresses that two arguments which have the same attackers and are not conflicting with each other can be merged to a single argument.

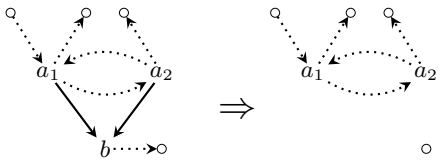


**Fig. 2.** The pattern $P^{3C}_{\{a_1,a_2,b\}}$.

**Definition 24.** *Let* $F_{2to1} = (\{a, b\}, \emptyset)$ *be the core-AF. The* 2-to-1 pattern *is given by*

$$P^{2to1}_{\{a,b\}} = \{(F, F') \mid F \text{ is a lagging of } F_{2to1},\\ \{a\}^-_F = \{b\}^-_F, F' = \mathsf{M}(F, a, b)\}.$$

The pattern $P^{2to1}_{\{a,b\}}$ is illustrated in Figure 3.

All patterns presented so far can be generalized. We exemplify this by presenting the patterns 4-path, 4-cone, and 3-to-2, extending 3-path, 3-cone, and 2-to-1, respectively.

First, in the 4-path pattern the core-AF is the directed path $a \to b \to c \to d$ and, given that beside $(a, d)$ there are no further attacks among the core arguments, $a, c$ as well as $b, d$ can be merged as they will appear together in every (stable) extension. Loosely speaking we aim to replace in $F$ the 4-path $a \to b \to c \to d$ by the 2-path $m_{\{a,c\}} \to m_{\{b,d\}}$.

**Definition 25.** *Let* $F_{4P} = (\{a, b, c, d\}, \{(a, b), (b, c), (c, d)\})$ *be the core-AF. The* 4-path pattern *is given by*

$$P^{4P}_{\{a,b,c,d\}} = \{(F, F') \mid F \setminus \{(a, d)\} \text{ is a lagging of } F_{4P},\\ \{b, c, d\}^-_F = \{a, b, c\}, F' = \mathsf{M}(\mathsf{M}(F \setminus \{(b, c)\}, a, c), b, d)\}.$$

In the 4-cone pattern we consider three arguments in arbitrary attack relation, one of them being otherwise unattacked. Each stable extension contains at least one of the three arguments and we can remove any argument that is attacked by the three without attacking back.

**Definition 26.** *Let* $F_{4C} = (\{a_1, a_2, a_3, b\}, \{(a_1, b), (a_2, b), (a_3, b)\})$ *be the core-AF. The* 4-cone pattern *is given by*

$$P^{4C}_{\{a_1,a_2,a_3,b\}} = \{(F, F') \mid F \setminus \{(a_i, a_j) \mid i \neq j \in \{1, 2, 3\}\}\\ \text{is a lagging of } F_{4G}, \{a_3\}^-_F \subseteq \{a_1, a_2\}, F' = F \setminus \{b\}\}.$$

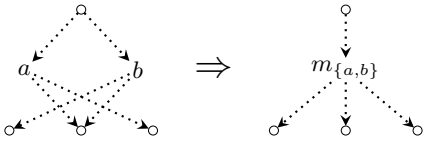


**Fig. 3.** The pattern $P^{2to1}_{\{a,b\}}$.

Finally, in the 3-to-2 pattern we consider three arguments $a_1, a_2, b$ that are attacked by the same arguments, and only $a_1$ and $a_2$ are conflicting. Each stable extension can only accept one of $a_1$ and $a_2$ but whenever accepting one of them also accepts $b$. Thus we can safely replace the three arguments by two merged arguments $m_{\{a_1,b\}}, m_{\{a_2,b\}}$.

**Definition 27.** *Let* $F_{3to2} = (\{a_1, a_2, b\}, \emptyset)$ *be the core-AF. The* 3-to-2 pattern *is given by*

$$P^{3to2}_{\{a_1,a_2,b\}} = \{(F, (A', R')) \mid F \setminus \{(a_1, a_2), (a_2, a_1)\} \text{ is a lagging of } F_{3to2},$$
$$\{a_1\}^-_F = \{a_2\}^-_F = \{b\}^-_F\},$$

*where* $A' = A_G \setminus \{a_1, a_2, b\} \cup \{m_{\{a_1,b\}}, m_{\{a_2,b\}}\}$ *and*

$$R' = R_G \cap (A' \times A') \cup \{(m_{\{a,b\}}, c) \mid a \in \{a_1, a_2\} \wedge ((a, c) \in R_G \vee (b, c) \in R_G)\} \cup$$
$$\{(c, m_{\{a_1,b\}}), (c, m_{\{a_2,b\}}) \mid c \in \{b\}^-_F\}).$$

*Patterns for stb semantics.* This concludes our replacement patterns for stable semantics. For preprocessing AFs, we apply these patterns recursively until no match for any of the patterns can be found. Notice that by the transitivity of the equivalence relation the recursive application of faithful replacement patterns is also equivalence preserving. For instance we simplify a 5-cycle by first applying the 3-path pattern and then the 3-loop pattern (cf. Fig. 1). Notice that, (a) when searching for matches of the pattern we only need to check whether a graph from a finite set of finite graphs appears as sub-AF and (b) with each replacement we delete either arguments or attacks, and thus the preprocessing is indeed in polynomial-time.

*Patterns for prf and com semantics.* While the path, 2to1, and 3to2 patterns are also *prf*-preserving and *com*-preserving (cf. Table 1) the remaining patterns, in general, are not. However, by a small modification, we can adapt the 3-loop pattern to work for *prf* as well as *com* as follows. Given an occurrence of a directed cycle of length 3, $a \to b \to c \to a$, where only $a$ is attacked from the outside, we can safely add a self-loop to $a$. In contrast to stable semantics we have to maintain that attack $(a, b)$, and cannot simply disregard $c$ but rather have to merge $a$ and $c$, i.e. we still can delete $c$ but for each attack $(c, x)$ in the original AF we have an attack $(a, x)$ in the simplified AF. This is formalized in the following replacement pattern. Notice that we can avoid using a merged argument $m_{\{a,c\}}$ as both $a$ and $c$ cannot appear in any extension.

**Definition 28.** *Let* $F_{3L} = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$ *be the core-AF. The* 3-loop pattern *for com and prf is given by*

$$P^{3L'}_{\{a,b,c\}} = \{(F, F') \mid F \setminus \{(a, a), (c, c)\} \text{ is a lagging of } F_{3L},$$
$$\{b, c\}^-_F \subseteq \{a, b, c\}, F' = (F \setminus \{c\}) \cup \{(a, x) \mid (c, x) \in R_F\}.$$

In order to adapt the 3-cone and 4-cone pattern for *prf*, one additionally requires that one of the arguments $a_i$ defends itself against all attackers. However, there is no such fix for *com* semantics.

**Theorem 3.** *The presented patterns are* $\sigma$*-preserving as depicted in Table 1.*

The proofs except for $P^{3to2}$ exploit Theorem 2 and the results of [5], following the same schema as the proof of Proposition 2. Finally, we note that further generalizations of our patterns are possible, e.g., extending the 3-loop pattern to 5-loop.

**Table 1.** $\sigma$-faithfulness of replacement patterns.

| | 3-path | 3-loop | 3-cone | 2to1 | 4-path | 4-cone | 3to2 |
|---|---|---|---|---|---|---|---|
| *stb* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *prf* | ✓ | (✓) | (✓) | ✓ | ✓ | (✓) | ✓ |
| *com* | ✓ | (✓) | × | ✓ | ✓ | × | ✓ |

## 4 Empirical Evaluation

We overview first empirical results of the potential of replacement patterns as an AF preprocessing approach in the context of *extension enumeration*. Our main goal was to investigate to which extent applying the patterns affects the AF solver running times.

In terms of AF semantics we overview results under stable and preferred semantics. As preprocessing, we applied all the presented patterns (cf. Table 1). For these first experiments, we implemented a software prototype for the application of the replacement patterns in a somewhat brute-force way. In particular, we encoded the search for a set of arguments to which a specific replacement pattern is applicable through an answer set programming (ASP) encoding, and iterated through all considered replacement patterns one-by-one until no applicable set of arguments was reported by the ASP solver Clingo (version 5.3.0). Note that this approach would require the ASP solver to *prove* in the end that a fixpoint is reached, i.e., that no replacements are applicable. To ensure the relatively fast termination of the preprocessing loop, we enforced a time limit of 5 seconds on each of the ASP solver calls, and terminated preprocessing for a particular pattern as soon as the ASP solver timed out. The experiments were run on Intel Xeon E5-2680 v4 2.4 GHz nodes with 128 GB RAM under CentOS 7. A per-instance timeout of 1800 seconds was enforced on each solver, with the preprocessing times included.

As benchmarks, we used a total of $440$ AFs based on the Watts-Strogatz model with the number of arguments $n \in \{500, 600, \ldots, 1500\}$ and parameters $k \in \{\lfloor \log_2(n) \rfloor - 1, \lfloor \log_2(n) \rfloor + 1\}$, $\beta \in \{0.1, 0.3, \ldots, 0.9\}$, and *probCycles* $\in \{0.1, 0.3, \ldots, 0.7\}$, generated using AFBenchGen2 [7] that was also employed in the 2017 ICCMA argumentation solver competition.

As for size reductions achieved on these instances, on average $11\%$ of arguments and $17\%$ of attacks were removed, with the maximum proportions of deleted arguments and attacks being $72\%$ and $80\%$, respectively.

Runtime comparisons with and without preprocessing under stable semantics are shown for ArgTools [15], Heureka [21], and CEGARTIX [10] in Figure 4, with the runtime of the original instance on the x-axis and the runtime of the preprocessed instance on the y-axis (with preprocessing time included). Applying the patterns has a strong effect on the runtimes of the native AF solvers ArgTools (Fig. 4 left) and Heureka (Fig. 4 center): some instances which originally took over 500 seconds for solving can now be solved in under 10 seconds. The number of timeouts also is reduced: from 141 to 134 for ArgTools and from 251 to 243 for Heureka. The contributions of preprocessing to the running times is fairly small even using the somewhat brute-force prototype implementation; the preprocessing overhead is from some seconds to around 20 seconds at most. This explains the increased running times on the easiest of the benchmark

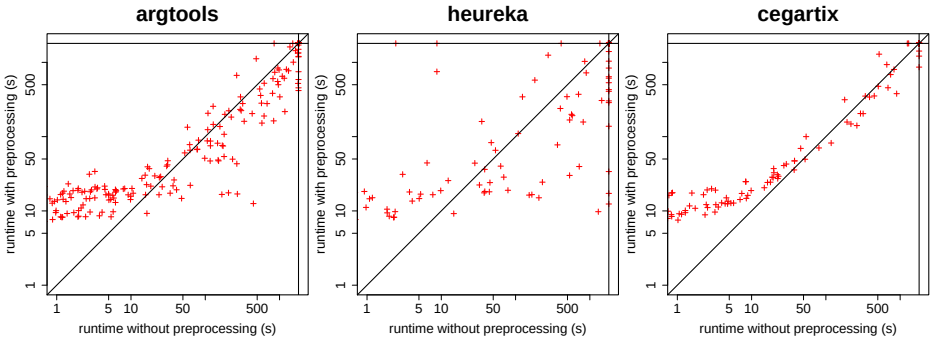


**Fig. 4.** Effect of preprocessing on runtimes of solvers under stable semantics.

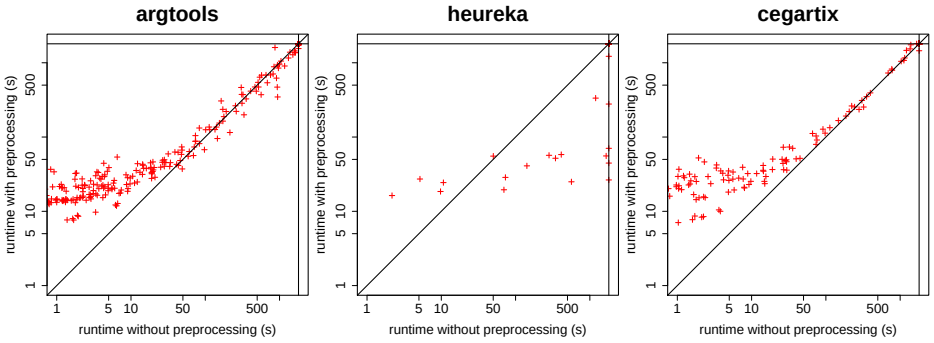


**Fig. 5.** Effect of preprocessing on runtimes of solvers under preferred semantics.

instances. The positive impact of preprocessing is evident on the harder (expectedly more interesting) benchmark instances. These results demonstrate the potential of AF preprocessing for improving the competitiveness of native AF solvers. In contrast, preprocessing appears to have no noticeable impact on the the SAT-based CEGARTIX system (Fig. 4 right).

Results for preferred semantics are shown in Fig. 5. Again, there seems to be no effect on CEGARTIX, and the effect on ArgTools is more modest when compared to the results for stable semantics. However, the effect on Heureka is similar to stable semantics, as it is able to solve instances which originally timed out without preprocessing, although Heureka seems to perform weakly on this particular set of instances under preferred semantics. As for size reductions, on average 6% of arguments and 9% of attacks were removed, with the maximum values being 62% and 72%, respectively.

## 5 Conclusions

In this paper, we introduced distinct preprocessing techniques for abstract argumentation frameworks which provide a solver-independent approach towards more efficient AF solving. Our formal framework of *replacement patterns* allows for identifying local

simplifications that are faithful w.r.t. standard semantics for AFs. We provided a suite of concrete replacement patterns and evaluated their impact with encouraging results especially for native AF solvers. So far we focused on equivalence-preserving preprocessing that allows for an easy reconstruction of all extensions of the original AF. We see even more potential for preprocessing in the context of credulous and skeptical acceptance, where faithfulness is required only in terms of a particular query argument; in that context, also the concept of S-equivalence of input/output AFs [2,16] deserves attention. Motivated by the first empirical results presented in this work, we are planning on engineering a fully-fledged stand-alone preprocessor, providing optimized implementations of applications of both the replacement patterns presented in this work as well as other forms of native AF preprocessing techniques. Furthermore, preprocessing rules (or restricted forms of them) may also be integrated into solvers for adding reasoning to the core search routine, which is another interesting topic for further work.

## Acknowledgments

This work was financially supported by Academy of Finland grants 276412 and 312662 (M.J. and A.N.), the Austrian Science Fund (FWF) grants P30168-N31 and I2854 (W.D. and S.W.), and University of Helsinki Doctoral Programme in Computer Science (A.N.).

## References

1. Atkinson, K., Baroni, P., Giacomin, M., Hunter, A., Prakken, H., Reed, C., Simari, G.R., Thimm, M., Villata, S.: Towards artificial argumentation. AI Magazine **38**(3), 25–36 (2017)
2. Baroni, P., Boella, G., Cerutti, F., Giacomin, M., van der Torre, L., Villata, S.: On the input/output behavior of argumentation frameworks. Artificial Intelligence **217**, 144–197 (2014)
3. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Engineering Review **26**(4), 365–410 (2011)
4. Baroni, P., Gabbay, D., Giacomin, M., van der Torre, L. (eds.): Handbook of Formal Argumentation. College Publications (2018)
5. Baumann, R., Dvořák, W., Linsbichler, T., Woltran, S.: A general notion of equivalence for abstract argumentation. In: Proc. IJCAI. pp. 800–806. ijcai.org (2017)
6. Cerutti, F., Gaggl, S.A., Thimm, M., Wallner, J.P.: Foundations of implementations for formal argumentation. IfCoLog Journal of Logic and its Applications **4**(8), 2623–2707 (2017)
7. Cerutti, F., Giacomin, M., Vallati, M.: Generating structured argumentation frameworks: AFBenchGen2. In: Proc. COMMA. Frontiers in Artificial Intelligence and Applications, vol. 287, pp. 467–468. IOS Press (2016)
8. Cerutti, F., Vallati, M., Giacomin, M.: Where are we now? State of the art and future trends of solvers for hard argumentation problems. In: Proc. COMMA. Frontiers in Artificial Intelligence and Applications, vol. 287, pp. 207–218. IOS Press (2016)
9. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77**(2), 321–358 (1995)
10. Dvořák, W., Järvisalo, M., Wallner, J.P., Woltran, S.: Complexity-sensitive decision procedures for abstract argumentation. Artificial Intelligence **206**(0), 53–78 (2014)

11. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. SAT. Lecture Notes in Computer Science, vol. 3569, pp. 61–75. Springer (2005)
12. Egly, U., Gaggl, S.A., Woltran, S.: Answer-set programming encodings for argumentation frameworks. Argument and Computation **1**(2), 147–177 (2010)
13. Gaggl, S.A., Linsbichler, T., Maratea, M., Woltran, S.: Introducing the Second International Competition on Computational Models of Argumentation. In: Proc. SAFA. CEUR Workshop Proceedings, vol. 1672, pp. 4–9. CEUR-WS.org (2016)
14. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Advanced preprocessing for answer set solving. In: Proc. ECAI. Frontiers in Artificial Intelligence and Applications, vol. 178, pp. 15–19. IOS Press (2008)
15. Geilen, N., Thimm, M.: Heureka: A general heuristic backtracking solver for abstract argumentation. In: Proc. TAFA. Lecture Notes in Computer Science, vol. 10757, pp. 143–149. Springer (2017)
16. Giacomin, M., Linsbichler, T., Woltran, S.: On the functional completeness of argumentation semantics. In: Proc. KR. pp. 43–52. AAAI Press (2016)
17. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. Journal of Artificial Intelligence Research **53**, 127–168 (2015)
18. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Proc. IJCAR. Lecture Notes in Computer Science, vol. 7364, pp. 355–370. Springer (2012)
19. Korhonen, T., Berg, J., Saikko, P., Järvisalo, M.: MaxPre: An extended MaxSAT preprocessor. In: Proc. SAT. Lecture Notes in Computer Science, vol. 10491, pp. 449–456. Springer (2017)
20. Lonsing, F., Bacchus, F., Biere, A., Egly, U., Seidl, M.: Enhancing search-based QBF solving by dynamic blocked clause elimination. In: Proc. LPAR-20. Lecture Notes in Computer Science, vol. 9450, pp. 418–433. Springer (2015)
21. Nofal, S., Atkinson, K., Dunne, P.E.: Looking-ahead in backtracking algorithms for abstract argumentation. International Journal of Approximate Reasoning **78**, 265–282 (2016)
22. Thimm, M., Villata, S.: The first international competition on computational models of argumentation: Results and analysis. Artificial Intelligence **252**, 267–294 (2017)